NASA/CR-1998-207682
ICASE Report No. 98-20

Three-Dimensional High-Lift Analysis Using a Parallel Unstructured Multigrid Solver

Dimitri J. Mavriplis
ICASE, Hampton, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

May 1998

THREE-DIMENSIONAL HIGH-LIFT ANALYSIS USING A PARALLEL UNSTRUCTURED MULTIGRID SOLVER

DIMITRI J. MAVRIPLIS *

Abstract. A directional implicit unstructured agglomeration multigrid solver is ported to shared and distributed memory massively parallel machines using the explicit domain-decomposition and message-passing approach. Because the algorithm operates on local implicit lines in the unstructured mesh, special care is required in partitioning the problem for parallel computing. A weighted partitioning strategy is described which avoids breaking the implicit lines across processor boundaries, while incurring minimal additional communication overhead. Good scalability is demonstrated on a 128 processor SGI Origin 2000 machine and on a 512 processor CRAY T3E machine for reasonably fine grids. The feasibility of performing large-scale unstructured grid calculations with the parallel multigrid algorithm is demonstrated by computing the flow over a partial-span flap wing high-lift geometry on a highly resolved grid of 13.5 million points in approximately 4 hours of wall clock time on the CRAY T3E.

Key words. parallel, multigrid, unstructured, Navier-Stokes

Subject classification. Applied and Numerical Mathematics

1. Introduction. The computation of three-dimensional high-lift flows constitutes one of the most challenging steady-state aerodynamic analysis problems today. Three-dimensional high-lift is typically characterized by complicated geometries, involving flaps, slats, and hinge fairings, in addition to very complex flow physics which must be captured adequately in order to provide a useful predictive capability for the design process.

Due to the complicated flow physics, Reynolds-averaged Navier-Stokes methods appear to be best suited for such problems in the long run, despite their higher computational requirements [20]. Such methods require the use of highly stretched grids in the thin boundary-layer and wake regions, as well as a large overall number of grid points to adequately resolve the flow physics. Three-dimensional high-lift Navier-Stokes simulations have been performed with block-structured and overset structured grid methods. However, the large grid resolution requirements strain the limits of current-day supercomputers. Furthermore, the complicated geometries associated with high-lift problems result in excessive grid generation time.

Unstructured grid techniques offer the potential for greatly reducing the grid generation time associated with such problems. Several unstructured grid generation packages suitable for viscous flow solvers have been demonstrated recently [18, 35, 8, 9]. Furthermore, unstructured mesh approaches enable the use of adaptive meshing techniques which hold great promise for increasing solution accuracy at minimal additional computational cost. On the other hand, unstructured mesh solvers require significantly higher computational resources than their structured grid counterparts. In particular, the large memory requirements of unstructured mesh solvers have made three-dimensional high-lift analysis, which already strains the capabilities of structured grid solvers, impractical with unstructured grids.

*ICASE, Mail Stop 403, NASA Langley Research Center, Hampton, Virginia 23681-2199, dimitri@icase.edu. This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

One of the often overlooked advantages of unstructured mesh approaches is their superior scalability on massively parallel computer architectures. In contrast with block-structured or overset-mesh methods, the data-structures used in unstructured mesh methods are homogeneous across the entire computational domain, thus enabling near perfect load-balancing and good overall scalability on parallel machines, even for small problems on large numbers of processors.

While the cost of specialized memory used in vector computer architectures has changed very little over the last several years, the cost of commodity memory used in parallel microprocessor-based machines has dropped dramatically, leading to the appearance of parallel machines with large shared or distributed memory systems. This development raises the possibility of performing large-scale unstructured-mesh computations suitable for high-lift analysis on such machines. The purpose of this paper is to demonstrate the feasibility of performing such calculations by porting a previously developed low-memory directional-implicit agglomeration-multigrid algorithm [11, 12] to modern parallel computer architectures.

2. Base Solver . The Reynolds averaged Navier-Stokes equations are discretized by a finite-volume technique on meshes of mixed element types which may include tetrahedra, pyramids, prisms, and hexahedra. In general, prismatic elements are used in the boundary layer and wake regions, while tetrahedra are used in the regions of inviscid flow. All elements of the grid are handled by a single unifying edge-based data-structure in the flow solver [14].

The governing equations are discretized using a central difference finite-volume technique with added matrix-based artificial dissipation. The matrix dissipation approximates a Roe Rieman-solver based upwind scheme [22], but relies on a biharmonic operator to achieve second-order accuracy, rather than on a gradient-based extrapolation strategy [11]. The thin-layer form of the Navier-Stokes equations is employed in all cases, and the viscous terms are discretized to second-order accuracy by finite-difference approximation. For multigrid calculations, a first-order discretization is employed for the convective terms on the coarse grid levels for all cases.

The basic time-stepping scheme is a three-stage explicit multistage scheme with stage coefficients optimized for high frequency damping properties [32], and a CFL number of 1.8. Convergence is accelerated by a local block Jacobi preconditioner, which involves inverting a 5×5 matrix for each vertex at each stage [21, 15, 16, 17]. A low-Mach number preconditioner [36, 29, 27] is also implemented. This is imperative for high-lift flows which may contain large regions of low Mach number flow particularly on the lower surfaces of the wing. The low-Mach number preconditioner is implemented by modifying the dissipation terms in the residual as described in [11], and then taking the corresponding linearization of these modified terms into account in the Jacobi preconditioner, a process sometimes referred to as *preconditioning*^{2} [11, 30].

The single equation turbulence model of Spalart and Allmaras [25] is utilized to account for turbulence effects. This equation is discretized and solved in a manner completely analogous to the flow equations, with the exception that the convective terms are only discretized to first-order accuracy.

3. Directional-Implicit Multigrid Algorithm . An agglomeration multigrid algorithm [14, 6, 24] is used to further enhance convergence to steady-state. In this approach, coarse levels are constructed by fusing together neighboring fine grid control volumes to form a smaller number of larger and more complex control volumes on the coarse grid. While agglomeration multigrid delivers very fast convergence rates for inviscid flow problems, the convergence obtained for viscous flow problems remains much slower, even when employing preconditioning techniques as described in the previous section. This slowdown is mainly due to the large degree of grid anisotropy in the viscous regions. Directional smoothing and coarsening techniques [11, 12] can be used to overcome this aspect-ratio induced stiffness.

Directional smoothing is achieved by constructing lines in the unstructured mesh along the direction of strong coupling (i.e. normal to the boundary layer) and solving the implicit system along these lines using a tridiagonal line solver. A weighted graph algorithm is used to construct the lines on each grid level, using edge weights based on the stencil coefficients for a scalar convection equation. This algorithm produces lines of variable length. In regions where the mesh becomes isotropic, the length of the lines reduces to zero (one vertex, zero edges), and the preconditioned explicit scheme described in the previous section is recovered. An example of the set of lines constructed from the two-dimensional unstructured grid in Figure 3.1 is depicted in Figure 3.2.

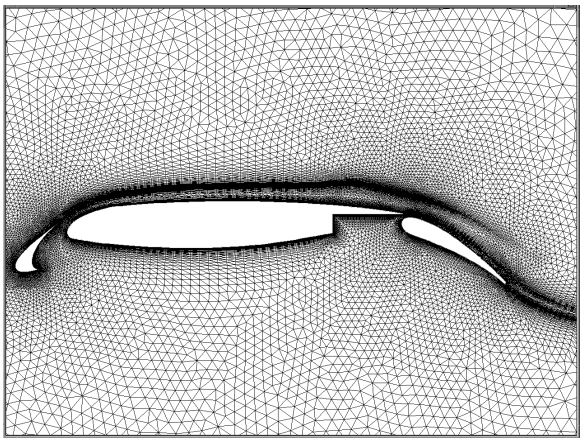

FIG. 3.1. *Unstructured Grid for three-element airfoil; Number of Points = 61,104, Wall Resolution* $= 10^{-6}$ *chords*

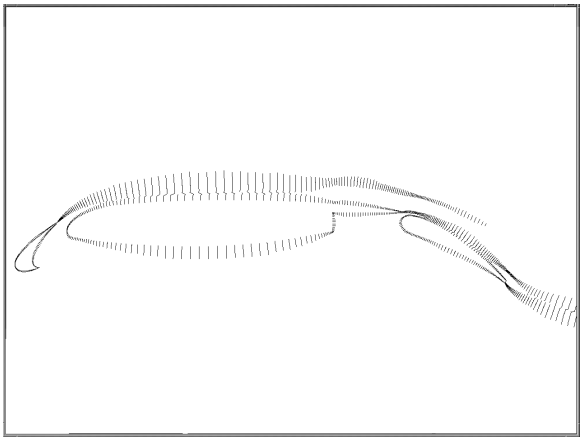

FIG. 3.2. *Directional Implicit Lines Constructed on Grid of Figure 3.1 by Weighted Graph Algorithm*

In addition to using a directional smoother, the agglomeration multigrid algorithm must be modified to take into account the effect of mesh stretching. The unweighted agglomeration algorithm which groups together all neighboring control volumes for a given fine grid vertex [14] is replaced with a weighted coarsening algorithm which only agglomerates the neighboring control volumes which are the most strongly connected to the current fine grid control volume, as determined by the same edge weights used in the line construction algorithm. This effectively results in semi-coarsening type behavior in regions of large mesh stretching, and regular coarsening in regions of isotropic mesh cells. In order to maintain favorable coarse grid complexity, an

aggressive coarsening strategy is used in anisotropic regions, where for every retained coarse grid point, three fine grid control volumes are agglomerated, resulting in an overall complexity reduction of 4:1 for the coarser levels in these regions, rather than the 2:1 reduction typically observed for semi-coarsening techniques. In isotropic regions an 8:1 coarsening ratio is obtained. However, since most of the mesh points reside in the boundary layer regions, the overall coarsening ratios achieved between grid levels is only slightly higher than 4:1. An example of the first directionally agglomerated level on a two-dimensional mesh is depicted in Figure 3.3, where the aggressive agglomeration normal to the boundary layer is observed.

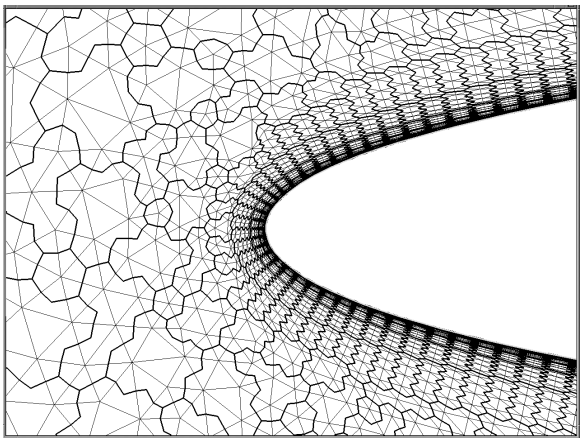

FIG. 3.3. *First Agglomerated Multigrid Level for Two-Dimensional Unstructured Grid Illustrating 4:1 Directional Coarsening in Boundary Layer Region*

4. Parallel Computer Architectures . For the purposes of this paper, parallel computer architectures can be classified by the configuration of their memory systems. These include:

- Multi-processors with uniform shared-memory access
- Cache-based processors with uniform (out-of-cache) shared-memory access
- Cache-based processors with non-uniform (out-of-cache) shared-memory access
- Cache-based processors with distributed memory

In the following we consider the computational model of an unstructured grid solver as an edge-based loop updating vertex-based values. For shared-memory architectures, parallelization can be achieved simply by reordering and grouping the edges of the loop, whereas for distributed-memory systems, explicit domain decomposition and message passing must be performed. Traditionally, only vector machines have been available with uniform shared-memory access. On these machines, vectorization is achieved by sorting the edges of the mesh into groups (colors), such that within each group no two edges access the same vertex. This enables vectorization of the edge loop over each color. Each color can then itself be split into multiple sub-groups, each of which is assigned to a particular processor. This approach is simple to implement and has been shown to work well on moderate numbers of processors, delivering speedups of up to 13 on 16 processors of the CRAY C90 [13]. Since each edge color extends across the entire mesh, this approach is only effective when a uniform memory access is available.

In the case of cache-based processors, locality becomes important, and alternative ordering strategies such as those suggested in [7] are required. In these methods, vertices and edges are reordered for locality, and separated into groups which can then be processed in parallel. This strategy can be used on cache-based machines with uniform shared-memory access such as SGI Power Challenge architectures, as well as

on cache-based machines with non-uniform shared-memory access, such as SGI Origin 2000 architectures. In the later case, the memory architecture is shared, in that it is globally addressable, but the access is non-uniform, in that it is physically distributed, and references to off-processor memory locations are more expensive than references to local processor memory locations. For such memory architectures, (which can be viewed as an additional cache level), data locality is even more important for performance than for cache-based uniform shared-memory architectures. The above strategy is attractive because it can be implemented through a relatively simple run-time data reordering operation, for any solver which already makes use of edge groupings (as for previous vectorized codes). This approach works relatively well for small numbers of processors, but has been found to scale poorly for larger numbers of processors.

For distributed-memory architectures, memory is not globally addressable, and an explicit domain decomposition/message passing implementation must be performed. Unstructured mesh domain decomposition or partitioning can be accomplished by a number of well documented and available partitioning strategies, which attempt to maintain load balancing and reduce communication volume by producing cuts that minimize the total number of edges which are intersected by the partition separators [19, 4, 1]. The partitioned data must also be optimized for cache efficiency by reordering the vertices and edges in each partition using a bandwidth minimization technique such as the Cuthill McKee [2] reordering algorithm applied locally in each partition.

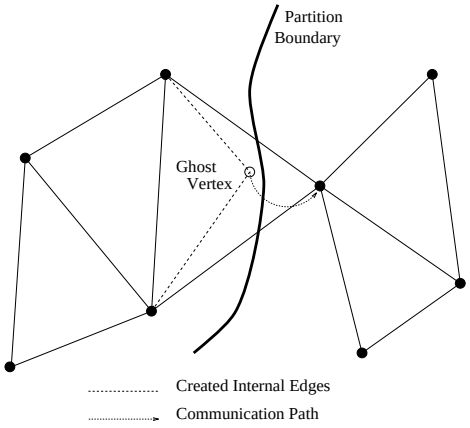

FIG. 4.1. *Illustration of Creation of Internal Edges and Ghost Points at Inter-processor Boundaries*

Inter-processor communication is generated by mesh edges which straddle two adjacent mesh partitions. Such edges are assigned to one of the partitions, and a "ghost vertex" is constructed in this partition which corresponds to the vertex originally accessed by the edge in the adjacent partition, as depicted in Figure 4.1 During a residual evaluation, the fluxes are computed along edges and accumulated to the vertices. The flux contributions accumulated at the ghost vertices must then be added to the flux contributions at their corresponding physical vertex locations in neighboring partitions in order to obtain the complete residual at these points. This phase incurs interprocessor communication. A standard technique to reduce the latency associated with this communication consists of packing all messages to be transmitted between adjacent pairs of communicating processors into a single large message buffer, which is then transmitted and unpacked on the receiving processor [3, 33, 1].

The explicit domain decomposition and message passing approach can also be implemented on shared-memory machines. In fact, for large numbers of processors, this approach generally scales more favorably than the simple reordering strategies discussed previously. This is largely due to the fact that reordering

based techniques on shared-memory machines implicitly generate communication at the loop level, each time an off-processor or non-local memory reference is encountered. In the explicit approach, duplicate remote memory references are omitted through the use of local ghost points and pre-computed optimal communication patterns, which are invoked only at critical locations in the program, such as at the end of a residual evaluation.

The implementation described in the remainder of this paper makes use of the explicit message passing approach using the MPI library. This makes for a portable program which can be run on distributed-memory parallel machines such as the CRAY T3E, as well as shared-memory machines such as the SGI Origin 2000. For shared-memory architectures, the isolated communication routines, which make use of the MPI libraries, could be replaced with simple copies of data from local to remote memory locations, although this has not been attempted. (The *"shmem"* routines on the CRAY T3E and SGI Origin 2000 also provide a similar functionality, although this has not been exploited in the present implementation, in the interest of portability).

While the flow solver implementation does not make use of the globally addressable memory capability of shared-memory machines such as the SGI Origin 2000, this feature has been instrumental in enabling the execution of the various pre-processing operations which are required by the multigrid algorithm prior to the flow solution phase. In particular, the implicit line construction and multigrid agglomeration algorithms, as well as the mesh partitioning procedure [4], require little overall cpu time, but significant memory resources. Additionally, these routines involve considerable amount of logic, and their parallelization using explicit message passing is a relatively involved task. The shared-memory architecture of the SGI Origin 2000 enables the execution of these routines using a single processor, but accessing large portions of the entire 128 cpu machine memory (36 Gbytes). These partitioned pre-processed results were then employed as input data for the flow solver running either on the SGI Origin 2000, or the CRAY T3E. For increasingly massively parallel applications, the parallel implementation of these pre-processing applications will eventually be required. However, the complexity of this task should be substantially reduced using the shared-memory paradigm.

5. Parallel Implementation. Distributed-memory explicit message-passing parallel implementations of unstructured mesh solvers have been discussed extensively in the literature [13, 1, 34]. In this section we focus on the non-standard aspects of the present implementation which are particular to the directional-implicit agglomeration multigrid algorithm.

In the multigrid algorithm, the vertices on each grid level must be partitioned across the processors of the machine. Since the mesh levels of the agglomeration multigrid algorithm are fully nested, a partition of the fine grid could be used to infer a partition of all coarser grid levels. While this would minimize the communication in the inter-grid transfer routines, it affords little control over the quality of the coarse grid partitions. Since the amount of intra-grid computation on each level is much more important than the inter-grid computation between each level, it is essential to optimize the partitions on each grid level rather than between grid levels. Therefore, each grid level is partitioned independently. This results in unrelated coarse and fine grid partitions. In order to minimize inter-grid communication, the coarse level partitions are renumbered such that they are assigned to the same processor as the fine grid partition with which they share the most overlap.

For each partitioned level, the edges of the mesh which straddle two adjacent processors are assigned to one of the processors, and a "ghost vertex" is constructed in this processor, which corresponds to the vertex originally accessed by the edge in the adjacent processor (c.f. Figure 4.1). During a residual evaluation, the

fluxes are computed along edges and accumulated to the vertices. The flux contributions accumulated at the ghost vertices must then be added to the flux contributions at their corresponding physical vertex locations in order to obtain the complete residual at these points. This phase incurs interprocessor communication. In an explicit (or point implicit) scheme, the updates at all points can then be computed without any interprocessor communication once the residuals at all points have been calculated. The newly updated values are then communicated to the ghost points, and the process is repeated.

The use of line-solvers can lead to additional complications for distributed-memory parallel implementations. Since the classical tridiagonal line-solve is an inherently sequential operation, any line which is split between multiple processors will result in processors remaining idle while the off-processor portion of their line is computed on a neighboring processor. However, the particular topology of the line sets in the unstructured grid permit a partitioning the mesh in such a manner that lines are completely contained within an individual processor, with minimal penalty (in terms of processor imbalance or additional numbers of cut edges). This can be achieved by using a weighted-graph-based mesh partitioner such as the CHACO partitioner [4]. Weighted graph partitioning strategies attempt to generate balanced partitions of sets of weighted vertices, and to minimize the sum of weighted edges which are intersected by the partition boundaries.

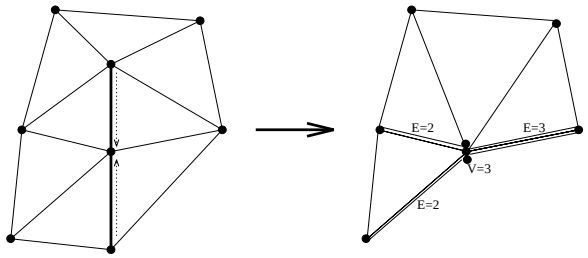

FIG. 5.1. *Illustration of Line Edge Contraction and Creation of Weighted Graph for Mesh Partitioning; V and E Values Denote Vertex and Edge Weights Respectively*

In order to avoid partitioning across implicit lines, the original unweighted graph (set of vertices and edges) which defines the unstructured mesh is contracted along the implicit lines to produce a weighted graph. Unity weights are assigned to the original graph, and any two vertices which are joined by an edge which is part of an implicit line are then merged together to form a new vertex. Merging vertices also produce merged edges as shown in Figure 5.1, and the weights associated with the merged vertices and edges are taken as the sum of the weights of the constituent vertices or edges. The contracted weighted graph is then partitioned using the partitioner described in [4, 5], and the resulting partitioned graph is then de-contracted, i.e. all constituent vertices of a merged vertex are assigned the partition number of that vertex. Since the implicit lines reduce to a single point in the contracted graph, they can never be broken by the partitioning process. The weighting assigned to the contracted graph ensures load balancing and communication optimization of the final uncontracted graph in the partitioning process.

The CHACO partitioner [4] provides options for various partitioning algorithms. We use the multi-level bisection partitioning option exclusively [5] in this work since it provides good partitions at low computational cost. As an example, the two dimensional mesh in Figure 3.1, which contains the implicit lines depicted in Figure 3.2, has been partitioned both in its original unweighted uncontracted form, and by the graph contraction method described above. Figure 5.2 depicts the results of both approaches for a 32-way partition. The unweighted partition contains 4760 cut edges (2.6 % of total), of which 1041 are line edges (also 2.6 %

of total), while the weighted partition contains no intersected line edges and a total of 5883 cut edges (3.2 % of total), i.e. a 23% increase over the total number of cut edges in the non-weighted partition.

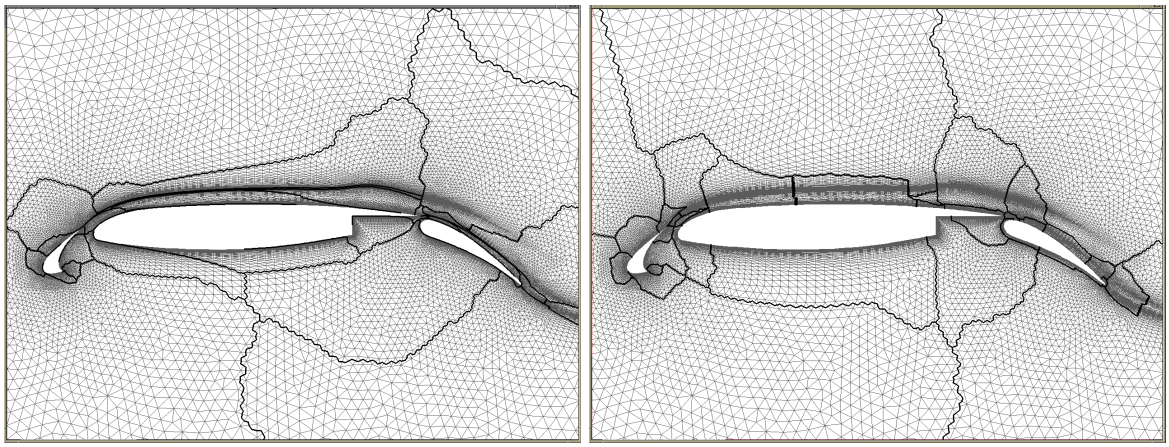

FIG. 5.2. *Comparison of Unweighted (above) and Weighted (below) 32-Way Partition of Two-Dimensional Mesh*

Although the resulting partitions are balanced in terms of the number of vertices in each partition, no attempt is made to balance the number of line edges in each partition. In fact, most often there is great disparity in the number of line edges in the various partitions. Furthermore, it is not feasible to attempt to modify the vertex partition balance in order to account for the extra work incurred by the line-solves. This is due to the fact that each stage of a multi-stage time-step consists of a residual evaluation followed by a point or line-solve, with communication (and synchronization) occurring at the end of the residual evaluation and the point/line-solve. Thus, ideally, the partitions must be vertex balanced for the residual evaluation phase, but line balanced for the solution phase. Fortunately, the amount of work involved in a residual evaluation is much larger than that involved in the solution phase, and the additional work of a (block) line-solve versus a (block) point-solve is such that the line imbalance does not appreciably affect the overall computational efficiency.

6. Results. The scalability of the directional implicit multigrid algorithm is examined on an SGI Origin 2000 and a CRAY T3E machine. The SGI Origin 2000 machine contains 128 MIPS R10000 195 Mhz processors with 286 Mbytes of memory per processor, for an aggregate memory capacity of 36.6 Gbytes. The CRAY T3E contains 512 DEC Alpha 300 Mhz processors with 128 Mbytes of memory per processor, for an aggregate memory capacity of 65 Gbytes. All the cases reported in this section were run in dedicated mode.

The first case consists of a relatively coarse 177,837 point grid over a swept and twisted wing, constructed by extruding a two-dimensional grid over an RAE 2822 airfoil in the spanwise direction. Figure 6.1 illustrates the grid for this case along with the implicit lines used by the solution algorithm on the finest level. The grid contains hexahedra in the boundary layer and (spanwise) prismatic elements in regions of inviscid flow, and exhibits a normal spacing at the wing surface of 10^{-6} chords. Approximately 67% of the fine grid points are contained within an implicit line, and no implicit lines on any grid levels were intersected in the partitioning process for all cases. This case was run at a freestream Mach number of 0.1, an incidence of 2.31 degrees, and a Reynolds number of 6.5 million. Figure 6.2 illustrates the computed solution obtained on this grid as a set of density contours on the surface. The convergence of the directional implicit multigrid

algorithm is compared with that achieved by the explicit isotropic multigrid algorithm [14] on the equivalent two dimensional problem in Figure 6.3. The directional implicit multigrid algorithm is seem to be much more effective than the isotropic algorithm, reducing the residuals by twelve orders of magnitude over 600 multigrid W-cycles. In the three-dimensional case, the turbulence model was frozen after 200 multigrid cycles.

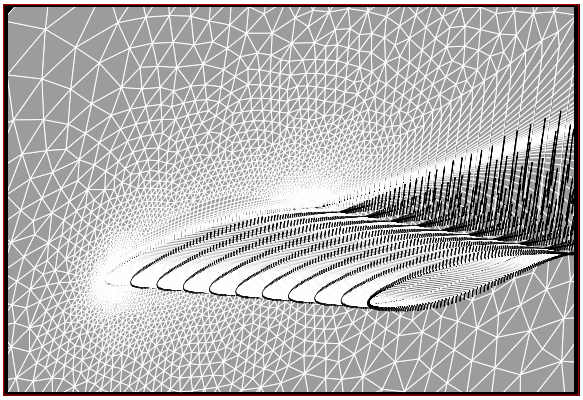

FIG. 6.1. *Unstructured Grid and Implicit Lines Employed for Computing Flow over Three-Dimensional Swept and Twisted RAE Wing*

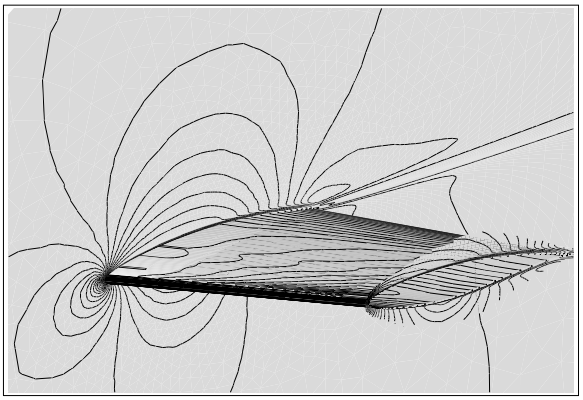

FIG. 6.2. *Computed Density Contours for Flow over Three-Dimensional Swept and Twisted RAE Wing. Mach = 0.1, Incidence = 2.31 degrees, Re number = 6.5 million*

The second test case involves a finer grid of 1.98 million points over an ONERA M6 wing. The grid was generated using the VGRID program [18]. A post-processing operation was employed to merge the tetrahedral elements in the boundary layer region into prisms [14]. The final grid contains 2.4 million prismatic elements and 4.6 million tetrahedral elements, and exhibits a normal spacing at the wall of 10^{-7} chords. Approximately 62% of the fine grid points are contained within an implicit line, and no implicit lines on any grid levels were intersected in the partitioning process for all cases. The freestream Mach number is 0.1, the incidence is 2.0 degrees, and the Reynolds number is 3 million. The convergence for this case is depicted in Figure 6.4, where the residuals are reduced by seven orders of magnitude over 600 multigrid W-cycles.

Figures 6.5 and 6.6 show the relative speedups achieved on the two target hardware platforms for the RAE wing case, while Figures 6.7 and 6.8 depict the corresponding results for the ONERA M6 wing case. For the purposes of these figures, perfect speedups were assumed on the lowest number of processors for which each case was run, and all other speedups are computed relative to this value. In all cases, timings were measured for the single grid (non-multigrid) algorithm, the multigrid algorithm using a V-cycle, and the multigrid algorithm using a W-cycle. Note that the best convergence rates, i.e. those displayed in Figures 6.3 and 6.4, are achieved using the W-cycle multigrid algorithm.

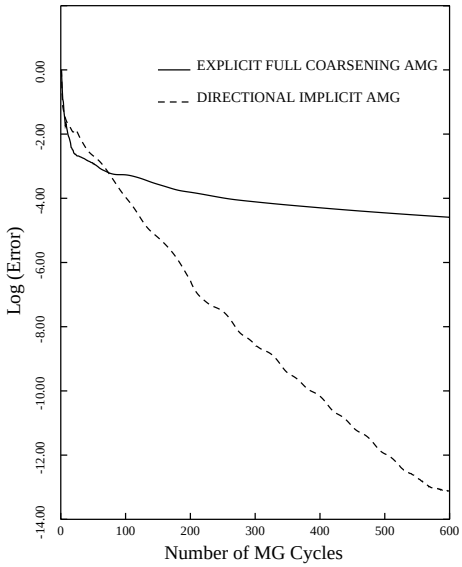

FIG. 6.3. *Comparison of Convergence Rate Achieved by Directional Implicit Agglomeration Multigrid versus Explicit Agglomeration Multigrid for Flow over Swept and Twisted Wing; Mach = 0.1, Incidence = 2.31 degrees, Re number = 6.5 million*

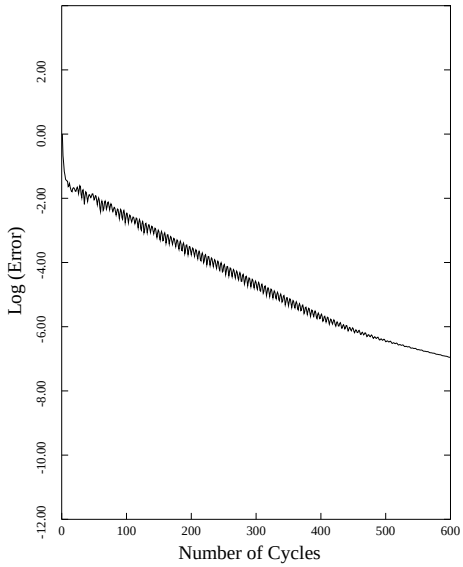

FIG. 6.4. *Multigrid Convergence Rate for 1.98 million Point Grid over ONERA M6 Wing; Mach = 0.1, Incidence = 2.0 degrees, Re number = 3 million.*

For the coarse RAE wing case, the results show good scalability up to moderate numbers of processors, while the finer ONERA M6 wing case shows good scalability up to the maximum number of processors on each machine, with only a slight drop-off at the higher numbers of processors. This is to be expected, since the relative ratio of computation to communication is higher for finer grids. This effect is also demonstrated by the superior scalability of the single grid algorithm versus the multigrid algorithms, and of the V-cycle multigrid algorithm over the W-cycle multigrid algorithm (i.e. the W-cycle multigrid algorithm performs additional coarse grid sweeps compared to the V-cycle algorithm). Note that for the RAE wing test case on 512 processors of the T3E, the fine grid contained only 348 vertices per processor, while the coarsest level contained a mere 13 points per processor. While the W-cycle algorithm suffers somewhat in computational performance for coarser grids on high processor counts, the parallel performance of the W-cycle improves substantially for finer grids. Numerically the most robust and efficient convergence rates are achieved using this cycle.

Figures 6.6 and 6.8 clearly illustrate the superior scalability on the T3E for these cases. In Tables 6.1 through 6.4, the timings per cycle and estimated computational rates are shown. These computational rates were estimated by running the same problem on a CRAY C90, and scaling the computational rates measured by the **hpm** command by the ratio of wall clock time on the target machine to the total cpu time on the CRAY C90. As the tables indicate, the individual processors of the Origin 2000 are up to 70% faster than those of the T3E. For example, on 64 processors, the ONERA M6 wing single grid case (a case which incurs little communication overhead) requires 10.2 seconds per cycle on the ORIGIN 2000, while the same case requires 17.1 seconds per cycle on 64 processors of the T3E.

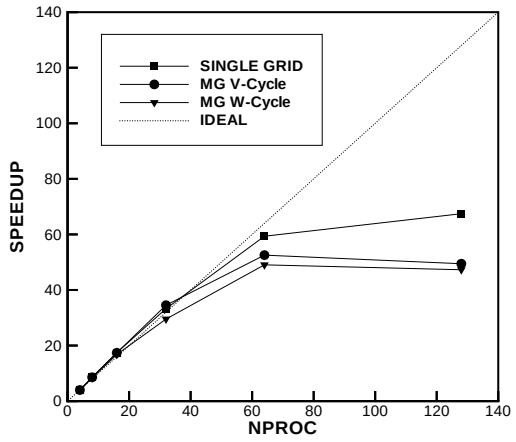

FIG. 6.5. *Observed Speedups for RAE Wing Case (177,837 grid points) on SGI Origin 2000*

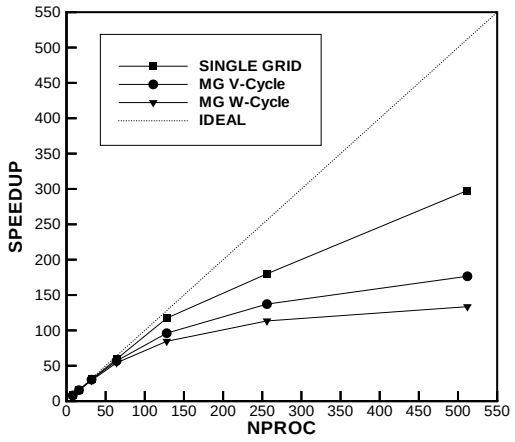

FIG. 6.6. *Observed Speedups for RAE Wing Case (177,837 grid points) on CRAY T3E*

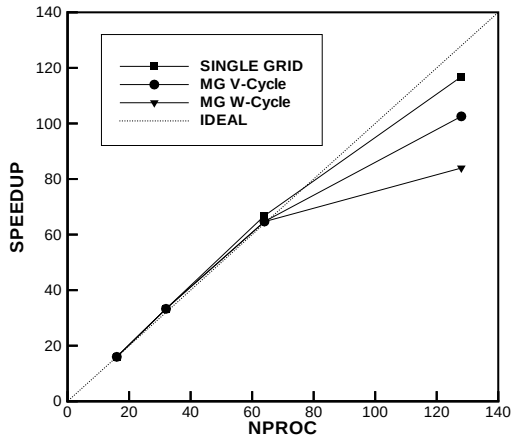

FIG. 6.7. *Observed Speedups for ONERA M6 Wing Case (1.98 million grid points) on SGI Origin 2000*

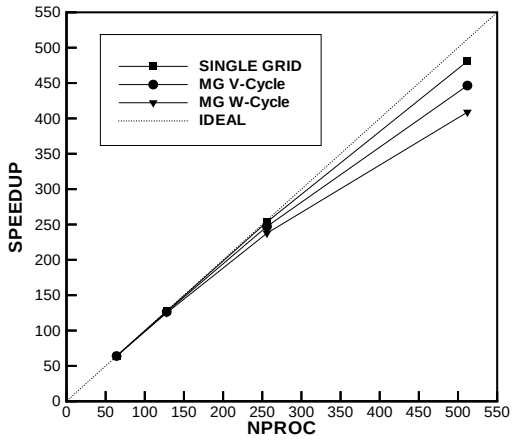

FIG. 6.8. *Observed Speedups for ONERA M6 Wing Case (1.98 million grid points) on CRAY T3E*

However, the better scalability demonstrated on the T3E is indicative of its lower latency and higher bandwidth communication capability. (Additionally, the newer T3E-1200 using 600Mhz processors should provide faster computational rates). In all cases, the fastest overall computational rates are achieved on the 512 processor configuration of the T3E. In this configuration, the RAE wing test case can be solved to machine zero in just 10 minutes (i.e. 600 W-cycles as per Figure 6.3). The 1.98 million point grid for the ONERA M6 wing case requires just 4.9 seconds per cycle on the 512 processor T3E, or 14.2 seconds per cycle on the 128 processor Origin 2000. From Figure 6.4 it can be deduced that a calculation to engineering accuracy can be performed in approximately 300 W-cycles, which requires 25 minutes on the T3E or 71 minutes on the Origin 2000.

SGI Origin 2000 Single Grid (1.98 million points)			
No. of Procs	Time/Cyc	Gflop/s	Speedup
16	44.6	1.27	1.0
32	21.24	2.67	2.10
64	10.18	5.56	4.38
128	5.82	9.73	7.66

TABLE 6.1

Timings and Estimated Computational Rates for Single Grid ONERA M6 Wing Case on SGI Origin 2000 Architecture

SGI Origin 2000 Multigrid W-Cycle (1.98 million points)			
No. of Procs	Time/Cyc	Gflop/s	Speedup
16	94.02	1.04	1.0
32	45.12	2.16	2.08
64	22.36	4.36	4.20
128	14.22	6.87	6.61

TABLE 6.2

Timings and Estimated Computational Rates for W-cycle Multigrid ONERA M6 Wing Case on SGI Origin 2000 Architecture

CRAY T3E Single Grid (1.98 million points)			
No. of Procs	Time/Cyc	Gflop/s	Speedup
64	17.13	3.30	1.0
128	8.60	6.57	1.99
256	4.32	13.07	3.96
512	2.28	24.8	7.51

TABLE 6.3

Timings and Estimated Computational Rates for Single Grid ONERA M6 Wing Case on CRAY T3E Architecture

CRAY T3E Multigrid W-Cycle (1.98 million points)			
No. of Procs	Time/Cyc	Gflop/s	Speedup
64	31.3	3.12	1.0
128	16.02	6.10	1.95
256	8.43	11.57	3.71
512	4.90	20.00	6.39

TABLE 6.4

Timings and Estimated Computational Rates for W-Cycle Multigrid ONERA M6 Wing Case on CRAY T3E Architecture

The next test case involves the computation of flow over a three-dimensional high-lift component geometry. The geometry consists of an unswept wing with a partial span flap, which has been the subject of both experimental and computational investigations [26, 23, 28]. The wing is mounted between two end walls, and the flap extends to the mid-span location. The flap deflection is 30 degrees, while the overall incidence of the geometry with respect to the undisturbed flow is 10 degrees. The freestream Mach number is 0.2, and the Reynolds number is 3.7 million. The flow over this geometry has been computed on an initial grid of 1.7 million points, and on a finer grid of 13.5 million points. The coarse grid exhibits a normal spacing at the wing surface of 10^{-6} chords, and the height of the cells in the boundary layer region follows a geometric progression with a growth rate of 1.2. This grid was generated using the VGRID program and originally contained 10 million tetrahedra, which were merged into 2.1 million prisms and 3.6 million tetrahedra through postprocessing. The finer grid was obtained through a uniform subdivision of the previous grid, using the mixed element adaptive meshing program described in [10], resulting in 17 million prisms, and 29 million tetrahedra. The high degree of resolution of this mesh is depicted in Figure 6.9. For both grids, approximately 70% of the vertices belong to implicit lines. The scalability of the coarser mesh is very similar to that displayed by the previous ONERA M6 wing case, and is therefore not reproduced here. This case was run for 600 W-cycles on 512 processors of the T3E, which required 3.76 seconds per W-cycle, or 38 minutes for the entire run, during which the residuals were reduced by just over five orders of magnitude. The fine grid case required 28 seconds per cycle on 512 processors of the T3E, or 236 minutes for a run of 220 W-cycles, during which the residual were reduced by just under four orders of magnitude, as depicted in Figure 6.11. At this

point, the fine grid achieves approximately the same level of convergence as the coarser grid for the same number of cycles, which is deemed sufficient for engineering accuracy. Although the computation has not been carried out further due to computational expense, an asymptotic slowdown similar to that observed on the coarser grid can be expected for the fine grid. For the fine grid case, the solver required approximately 69 Mbytes of memory per processor, or a total of 35.5 Gbytes of memory, which constitutes just over half the available memory on the machine. This total is approximately 50% higher than the memory requirements of the sequential solver on a per grid point basis, and is most likely due to imbalance between the processors, (dimensioning uses the maximum processor data-set size on all processors) as well as additional memory required for the ghost points and other message passing data-structures.

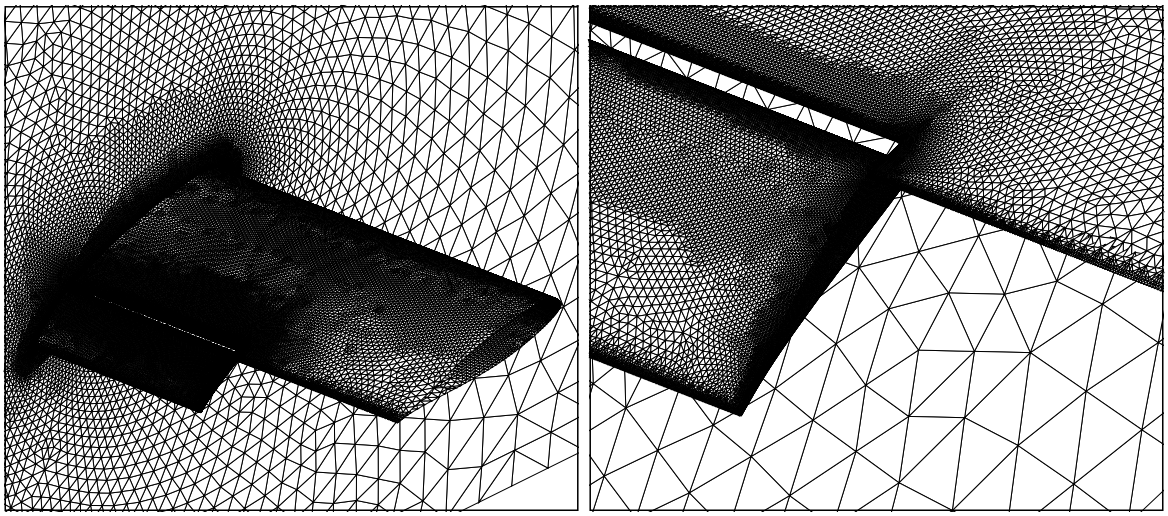

FIG. 6.9. *Illustration of 13.5 million point grid used to compute flow over partial span flap geometry.*

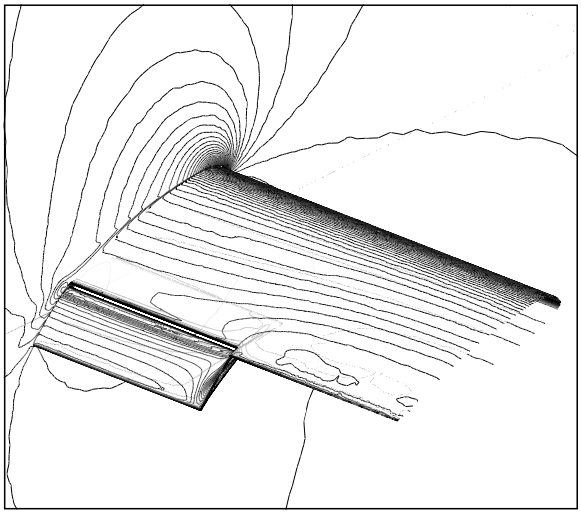

FIG. 6.10. *Computed Density Contours on 13.5 million point grid for flow over part-span flap geometry. Mach = 0.2, Incidence = 10 degrees, Reynolds number = 3.7 million.*

A summary of the resource requirements for the coarse and fine grid runs on the partial-span flap geometry is given in Table 6.5. Although the fine grid contains eight times the resolution of the coarse grid, the memory and cpu time required by the fine grid run are less than eight times that of the coarse grid run. This is due to the larger ratio of computation to communication in the fine grid case. Additionally, the fixed (non-data) memory requirement of the solver instruction set occupies 2.8 Mbytes per processor, which represents a smaller portion of the total memory requirements in the fine grid case.

The computed surface pressures for both grids are compared with experimental values at four spanwise locations in Figure 6.12. The coarse grid values tend to underpredict the suction peaks on the leading edge of the main wing. Much better agreement is obtained on the finer grid. The suction peaks are slightly overpredicted on the fine grid, although this is in agreement with previous numerical computations of this flow on block structured grids, which also over-predict the lift in these regions [28].

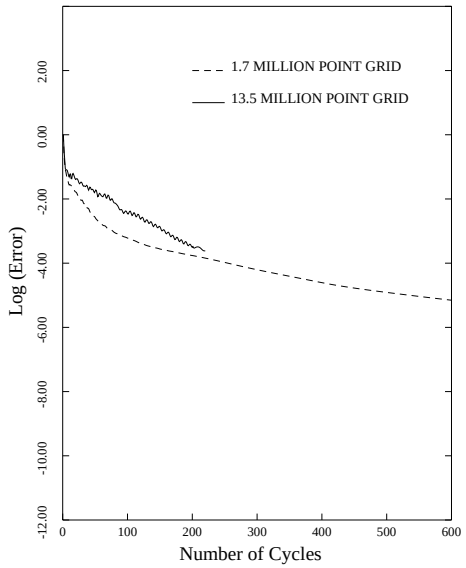

FIG. 6.11. *Multigrid Convergence Rates for Fine (13.5 million points) and coarse (1.7 million points) Grid over Part-Span Flap Geometry.*

The poor agreement of the coarse grid suction peaks with experimental values may be somewhat surprising, considering the better agreement obtained on previous unstructured computations performed with grids of similar overall size [14]. However, in these previous computations, the grids contained much lower boundary layer resolution, which was traded off for increased chordwise resolution. The boundary layer resolution employed in the current grids was derived from the resolution requirements established in two-dimensional high-lift computations [20, 31], and is believed to represent the necessary level for capturing detailed flow physics. On the other hand, no spanwise grid stretching was employed and thus a substantial savings in the total number of grid points with minimal accuracy degradation could probably be obtained by reducing the spanwise resolution. The fine grid calculation on the partial span flap geometry is intended to demonstrate the size of problems that can be attempted using this type of solver on present-day massively parallel computer architectures, rather than to define the number of grid points required for an accurate calculation of this type.

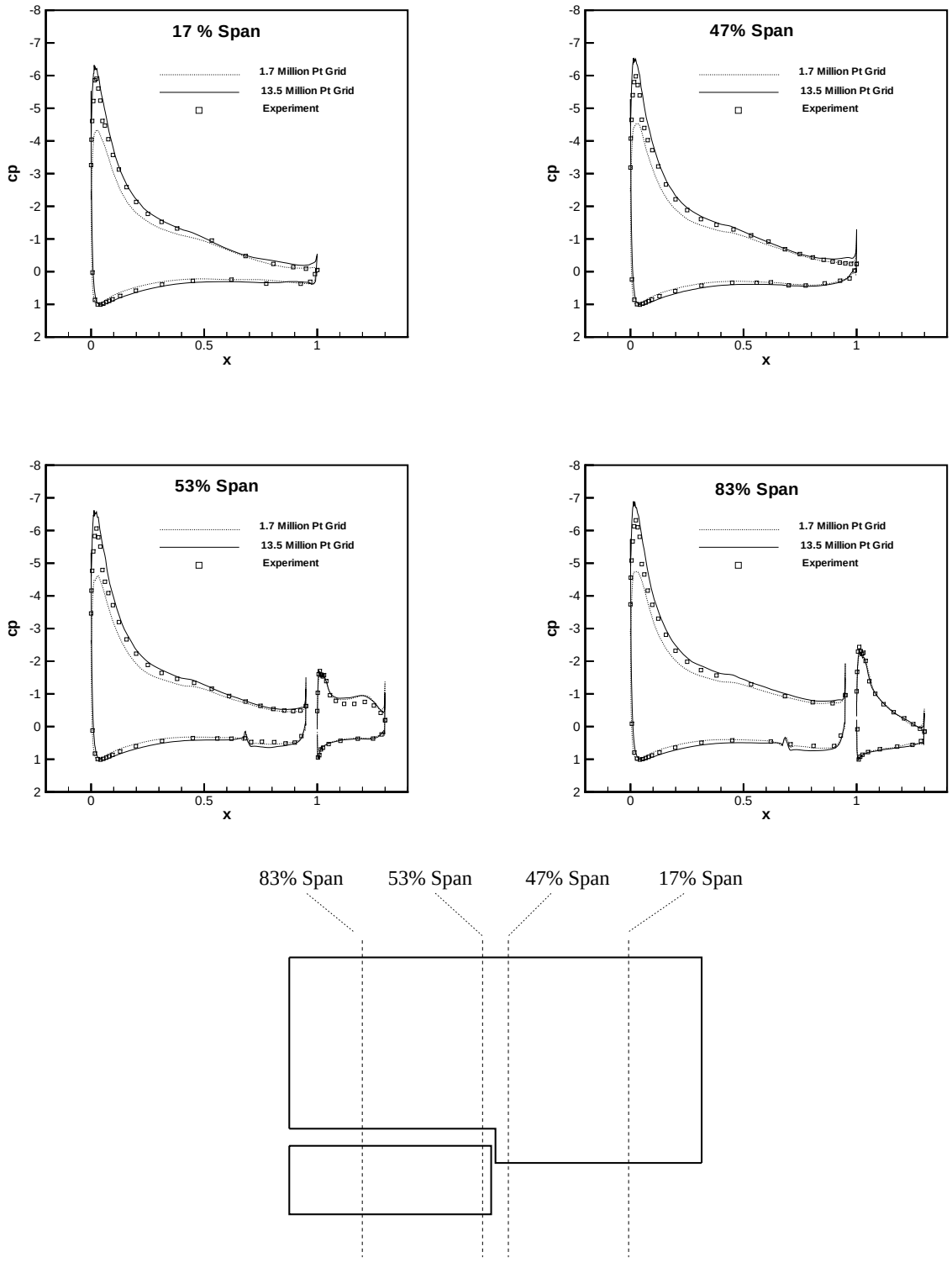

FIG. 6.12. *Comparison of Computed Surface Pressure Distribution with Experimental Results at Four Spanwise Locations on Partial Span Flap Geometry for Fine and Coarse Grids*

CRAY T3E (512 procs) Part-Span Flap Geometry (Multigrid W-Cycle)			
Grid	Mem Req.	Time/Cyc	Time to Sol.
1.7M	7.1 Gbytes	3.76	13 min
13.5M	35.5 Gbytes	27.75	236 min
Ratio (=8)	5.0	7.39	7.39

TABLE 6.5

Comparison of Coarse and Fine Grid Resource Requirements for Partial-Span Flap High-Lift Geometry. Time to Solution Represents Wall-Clock Time Required for 220 Multigrid W-Cycles in both Cases.

7. Conclusion. The combination of a low-memory rapidly converging directional implicit multigrid algorithm and massively parallel computer architectures with large shared or distributed memory has enabled the solution of very large unstructured grid problems in reasonable turnaround time. The 13.5 million grid point case described in this paper, which requires approximately half the available memory of a 512 processor T3E with 128 Mbytes per processor, is believed to be the largest unstructured grid problem attempted to date. An accurate solution of a full transport aircraft high-lift configuration will require several times more resolution. Such a calculation should be feasible on some of the largest currently available parallel machines. Furthermore, the relatively low cost of commodity memory employed by microprocessor-based parallel computers implies that it is now feasible to configure cost-effective mid-size machines with 32 to 64 processors and 1 Gbyte of memory per processor, thus enabling large scale unstructured grid computations suitable for high-lift analysis with overnight turnaround.

Future work will concentrate on improving the performance of the three dimensional multigrid algorithm which often produces sub-optimal convergence rates as compared to the equivalent two-dimensional algorithm [12]. Furthermore, many of the preprocessing operations required for the parallel computations, such as mesh partitioning, implicit line construction, and multigrid agglomeration are currently performed sequentially, and must eventually be parallelized. Finally, adaptive meshing techniques must be incorporated into the overall parallel solution strategy to take full advantage of the potential of unstructured meshes.

8. Acknowledgements. This work was made possible thanks to dedicated computer time donated by Cray Research, Eagan MN. Special thanks are due to David Whitaker for his assistance, as well as Jeff Dawson and Rob Vermeland. Acknowledgements are also due to Shayar Pirzadeh and Javier Garriz who provided the grids generated with the VGRID program.

REFERENCES

[1] *Special course on parallel computing in CFD*, May 1995. AGARD Report-807.

[2] E. CUTHILL AND J. MCKEE, *Reducing the band width of sparse symmetric matrices*, in Proc. ACM Nat. Conference, 1969, pp. 157–172.

[3] R. DAS, D. J. MAVRIPLIS, J. SALTZ, AND R. PONNUSAMY, *The design and implementation of a parallel unstructured Euler solver using software primitives.* AIAA Paper 92-0562, Jan. 1992.

[4] B. HENDRICKSON AND R. LELAND, *The Chaco user's guide: Version 2.0.* Tech. Rep. SAND94-2692, Sandia National Laboratories, Albuquerque, NM, July 1995.

[5] ——, *A multilevel algorithm for partitioning graphs*, in Proceedings Proc. Supercomputing '95, ACM, Dec. 1995.

[6] M. Lallemand, H. Steve, and A. Dervieux, *Unstructured multigridding by volume agglomeration: Current status*, Computers and Fluids, 21 (1992), pp. 397–433.

[7] R. Lohner, *Renumbering strategies for unstructured grid solvers operating on shared-memory cache-based parallel machines*, in Proceedings of the 13th AIAA CFD Conference, Snowmass, CO, June 1997, pp. 1015–1025. AIAA Paper 97-2045-CP.

[8] M. J. Marchant and N. P. Weatherill, *Unstructured grid generation for viscous flow simulations*, in Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, Pineridge Press, Apr. 1994, pp. 151–162. Proc. of the Fourth International Conference on Numerical Grid Generation in Computational Fluid Dynamics, Swansea, Wales, UK eds. N. P. Weatherill, P. R. Eiseman, J. Hauser, and J. F. Thompson.

[9] D. L. Marcum, *Generation of unstructured grids for viscous flow applications.* AIAA paper 95-0212, Jan. 1995.

[10] D. J. Mavriplis, *Adaptive meshing techniques for viscous flow calculations on mixed-element unstructured meshes.* AIAA paper 97-0857, Jan. 1997.

[11] ——, *Multigrid strategies for viscous flow solvers on anisotrpic unstructured meshes*, in Proceedings of the 13th AIAA CFD Conference, Snowmass, CO, June 1997, pp. 659–675. AIAA Paper 97-1952-CP.

[12] ——, *Directional agglomeration multigrid techniques for high-Reynolds number viscous flows.* AIAA paper 98-0612, Jan. 1998.

[13] D. J. Mavriplis, R. Das, J. Saltz, and R. E. Vermeland, *Implementation of a parallel unstructured Euler solver on shared and distributed memory machines*, The J. of Supercomputing, 8 (1995), pp. 329–344.

[14] D. J. Mavriplis and V. Venkatakrishnan, *A unified multigrid solver for the Navier-Stokes equations on mixed element meshes*, International Journal for Computational Fluid Dynamics, 8 (1997), pp. 247–263.

[15] E. Morano and A. Dervieux, *Looking for O(N) Navier-Stokes solutions on non-structured meshes*, in 6th Copper Mountain Conf. on Multigrid Methods, 1993, pp. 449–464. NASA Conference Publication 3224.

[16] C. Ollivier-Gooch, *Towards problem-independent multigrid cnvergence rates for unstructured mesh methods i: Inviscid and laminar flows*, in Proceedings of the 6th International Symposium on CFD, Lake Tahoe, NV, Sept. 1995.

[17] N. Pierce and M. Giles, *Preconditioning on stretched meshes.* AIAA paper 96-0889, Jan. 1996.

[18] S. Pirzadeh, *Viscous unstructured three-dimensional grids by the advancing- layers method.* AIAA paper 94-0417, Jan. 1994.

[19] A. Pothen, H. D. Simon, and K. P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 430–452.

[20] F. T. L. R. C. Potter and F. W. Spaid, *Requirements for effective high lift CFD*, in Proceedings of the 20th ICAS Congress, Sorrento, Italy, Sept. 1996, pp. 1479–1492. paper ICAS-96-2.7.1.

[21] K. Riemslagh and E. Dick, *A multigrid method for steady Euler equations on unstructured adaptive grids*, in 6th Copper Mountain Conf. on Multigrid Methods, NASA conference publication 3224, 1993, pp. 527–542.

[22] P. L. Roe, *Approximate Riemann solvers, parameter vectors and difference schemes*, J. Comp. Phys., 43 (1981), pp. 357–372.

[23] D. K. M. K. R. R. J. C. Ross and S. E. Rogers, *Navier-Stokes analysis of the flow about a flap*

edge. AIAA paper 95-0185, Jan. 1995.

[24] W. A. Smith, *Multigrid solution of transonic flow on unstructured grids*, in Recent Advances and Applications in Computational Fluid Dynamics, Nov. 1990. Proceedings of the ASME Winter Annual Meeting, Ed. O. Baysal.

[25] P. R. Spalart and S. R. Allmaras, *A one-equation turbulence model for aerodynamic flows*, La Recherche Aérospatiale, 1 (1994), pp. 5–21.

[26] B. L. Storms, T. T. Takahashi, and J. C. Ross, *Aerodynamic influence of a finite-span flap on a simple wing*. Paper to be presented at the SAE Aerotech Conference, Los Angeles, CA September 9-12, 1995.

[27] B. v. E. T. C. H. Tai and L. Mesaros, *Local preconditioning in a stagnation point*, in Proceedings of the 12th AIAA CFD Conference, San Diego, CA, June 1995, pp. 88–101. AIAA Paper 95-1654-CP.

[28] M. A. Takallu and K. R. Laflin, *Reynolds-averaged Navier-Stokes simulations of two partial-span flap wing experiments*. AIAA paper 98-0701, Jan. 1998.

[29] E. Turkel, *Preconditioning methods for solving the incompressible and low speed compressible equations*, J. Comp. Phys., 72 (1987), pp. 277–298.

[30] ——, *Preconditioning-squared methods for multidimensional aerodynamics*, in Proceedings of the 13th AIAA CFD Conference, Snowmass, CO, June 1997, pp. 856–866. AIAA Paper 97-2025-CP.

[31] W. O. Valarezo and D. J. Mavriplis, *Navier-Stokes applications to high-lift airfoil analysis*, AIAA J. of Aircraft, 23 (1995), pp. 457–688.

[32] B. van Leer, C. H. Tai, and K. G. Powell, *Design of optimally-smoothing multi-stage schemes for the Euler equations*. AIAA Paper 89-1933, June 1989.

[33] V. Venkatakrishnan, *Parallel implicit unstructured grid Euler solvers*, AIAA J., 32 (1994), pp. 1985–1991.

[34] ——, *Implicit schemes and parallel computing in unstructured grid CFD*, in VKI Lecture Series VKI-LS 1995-02, Mar. 1995.

[35] S. Ward and Y. Kallinderis, *Hybrid prismatic/tetrahedral grid generation for complex 3D geometries*. AIAA paper 93-0669, Jan. 1993.

[36] J. M. Weiss and W. A. Smith, *Preconditioning applied to variable and constant density time-accurate flows on unstructured meshes*. AIAA Paper 94-2209, June 1994.